EXHIBIT N

Oracle8*i*

Data Warehousing Guide

Release 2 (8.1.6)

December 1999

Part No. A76994-01

ORACLE

Data Warehousing Guide, Release 2 (8.1.6)

Part No. A76994-01

Primary Author: Paul Lane

Contributing Author: George Lumpkin

Contributors: Patrick Amor, Tolga Bozkaya, Karl Dias, Yu Gong, Ira Greenberg, Helen Grembowicz, John Haydu, Meg Hennington, Lilian Hobbs, Hakan Jakobsson, Jack Raitto, Ray Roccaforte, Andy Witkowski, Zia Ziauddin

Graphic Designer: Valarie Moore

3 Overview of Physical Design

4 Hardware and I/O

5 Parallelism and Partitioning

6 Indexes

7 Constraints

11 Extraction

12 Transportation

13 Transformation

14 Loading and Refreshing

19 Query Rewrite

2

Overview of Logical Design

This chapter tells how to design a data warehousing environment, and includes the following topics:

- Logical vs. Physical
- Create a Logical Design
- Data Warehousing Schemas

Logical vs. Physical

If you are reading this guide, it is likely that your organization has already decided to build a data warehouse. Moreover, it is likely that the business requirements are already defined, the scope of your application has been agreed upon, and you have a conceptual design. So now you need to translate your requirements into a system deliverable. In this step, you create the logical and physical design for the data warehouse and, in the process, define the specific data content, relationships within and between groups of data, the system environment supporting your data warehouse, the data transformations required, and the frequency with which data is refreshed.

The logical design is more conceptual and abstract than the physical design. In the ***logical design***, you look at the logical relationships among the objects. In the ***physical design***, you look at the most effective way of storing and retrieving the objects.

Your design should be oriented toward the needs of the end users. End users typically want to perform analysis and look at aggregated data, rather than at individual transactions. Your design is driven primarily by end-user utility, but the end users may not know what they need until they see it. A well-planned design allows for growth and changes as the needs of users change and evolve.

By beginning with the logical design, you focus on the information requirements without getting bogged down immediately with implementation detail.

Create a Logical Design

A logical design is a conceptual, abstract design. You do not deal with the physical implementation details yet; you deal only with defining the types of information that you need.

The process of logical design involves arranging data into a series of logical relationships called entities and attributes. An ***entity*** represents a chunk of information. In relational databases, an entity often maps to a table. An ***attribute*** is a component of an entity and helps define the uniqueness of the entity. In relational databases, an attribute maps to a column.

You can create the logical design using a pen and paper, or you can use a design tool such as Oracle Warehouse Builder or Oracle Designer.

While entity-relationship diagramming has traditionally been associated with highly normalized models such as online transaction processing (OLTP) applications, the technique is still useful in dimensional modeling. You just approach it differently. In dimensional modeling, instead of seeking to discover

atomic units of information and all of the relationships between them, you try to identify which information belongs to a central fact table(s) and which information belongs to its associated dimension tables.

One output of the logical design is a set of entities and attributes corresponding to fact tables and dimension tables. Another output of mapping is operational data from your source into subject-oriented information in your target data warehouse schema. You identify business subjects or fields of data, define relationships between business subjects, and name the attributes for each subject.

The elements that help you to determine the data warehouse schema are the model of your source data and your user requirements. Sometimes, you can get the source model from your company's enterprise data model and reverse-engineer the logical data model for the data warehouse from this. The physical implementation of the logical data warehouse model may require some changes due to your system parameters—size of machine, number of users, storage capacity, type of network, and software.

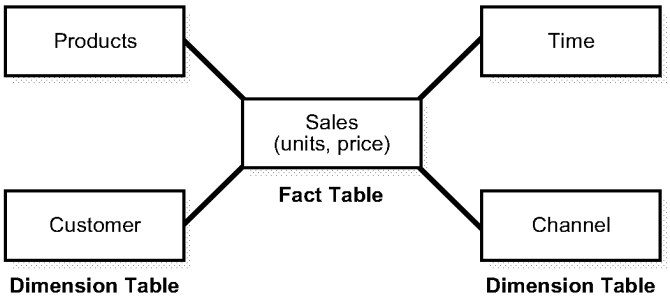
Data Warehousing Schemas

A schema is a collection of database objects, including tables, views, indexes, and synonyms. There are a variety of ways of arranging schema objects in the schema models designed for data warehousing. Most data warehouses use a dimensional model.

Star Schemas

The star schema is the simplest data warehouse schema. It is called a star schema because the diagram of a star schema resembles a star, with points radiating from a center. The center of the star consists of one or more fact tables and the points of the star are the dimension tables shown in Figure 2–1:

Figure 2–1 Star Schema



Unlike other database structures, in a star schema, the dimensions are denormalized. That is, the dimension tables have redundancy which eliminates the need for multiple joins on dimension tables. In a star schema, only one join is needed to establish the relationship between the fact table and any one of the dimension tables.

The main advantage to a star schema is optimized performance. A star schema keeps queries simple and provides fast response time because all the information about each level is stored in one row. See Chapter 16, "Schemas", for further information regarding schemas.

Note: Oracle recommends you choose a star schema unless you have a clear reason not to.

Other Schemas

Some schemas use third normal form rather than star schemas or the dimensional model.

Data Warehousing Objects

The following types of objects are commonly used in data warehouses:

- Fact tables are the central tables in your warehouse schema. Fact tables typically contain facts and foreign keys to the dimension tables. Fact tables represent data usually numeric and additive that can be analyzed and examined. Examples include Sales, Cost, and Profit.

- Dimension tables, also known as lookup or reference tables, contain the relatively static data in the warehouse. Examples are stores or products.

Fact Tables

A fact table is a table in a star schema that contains facts. A fact table typically has two types of columns: those that contain facts, and those that are foreign keys to dimension tables. A fact table might contain either detail-level facts or facts that have been aggregated. Fact tables that contain aggregated facts are often called ***summary tables***. A fact table usually contains facts with the same level of aggregation.

Values for facts or measures are usually not known in advance; they are observed and stored.

Fact tables are the basis for the data queried by OLAP tools.

Creating a New Fact Table

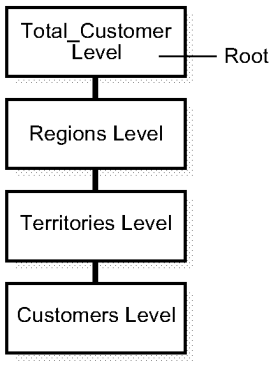
You must define a fact table for each star schema. A fact table typically has two types of columns: those that contain facts, and those that are foreign keys to dimension tables. From a modeling standpoint, the primary key of the fact table is usually a composite key that is made up of all of its foreign keys; in the physical data warehouse, the data warehouse administrator may or may not choose to create this primary key explicitly.

Facts support mathematical calculations used to report on and analyze the business. Some numeric data are dimensions in disguise, even if they seem to be facts. If you are not interested in a summarization of a particular item, the item may actually be a dimension. Database size and overall performance improve if you categorize borderline fields as dimensions.

Dimensions

A dimension is a structure, often composed of one or more hierarchies, that categorizes data. Several distinct dimensions, combined with measures, enable you to answer business questions. Commonly used dimensions are Customer, Product, and Time. Figure 2–2 shows some a typical dimension hierarchy.

Figure 2–2 Typical Levels in a Dimension Hierarchy



Dimension data is typically collected at the lowest level of detail and then aggregated into higher level totals, which is more useful for analysis. For example, in the Total_Customer dimension, there are four levels: Total_Customer, Regions, Territories, and Customers. Data collected at the Customers level is aggregated to the Territories level. For the Regions dimension, data collected for several regions such as Western Europe or Eastern Europe might be aggregated as a fact in the fact table into totals for a larger area such as Europe.

See Chapter 9, "Dimensions", for further information regarding dimensions.

Hierarchies

Hierarchies are logical structures that use ordered levels as a means of organizing data. A hierarchy can be used to define data aggregation. For example, in a Time dimension, a hierarchy might be used to aggregate data from the Month level to the Quarter level to the Year level. A hierarchy can also be used to define a navigational drill path and establish a family structure.

Within a hierarchy, each level is logically connected to the levels above and below it; data values at lower levels aggregate into the data values at higher levels. For example, in the Product dimension, there might be two hierarchies—one for product identification and one for product responsibility.

Dimension hierarchies also group levels from very general to very granular. Hierarchies are utilized by query tools, allowing you to drill down into your data to view different levels of granularity—one of the key benefits of a data warehouse.

When designing your hierarchies, you must consider the relationships defined in your source data. For example, a hierarchy design must honor the foreign key relationships between the source tables in order to properly aggregate data.

Hierarchies imposes a family structure on dimension values. For a particular level value, a value at the next higher level is its parent, and values at the next lower level are its children. These familial relationships allow analysts to access data quickly.

See Chapter 9, "Dimensions", for further information regarding hierarchies.

Levels Levels represent a position in a hierarchy. For example, a Time dimension might have a hierarchy that represents data at the Month, Quarter, and Year levels. Levels range from general to very specific, with the root level as the highest, or most general level. The levels in a dimension are organized into one or more hierarchies.

Level Relationships Level relationships specify top-to-bottom ordering of levels from most general (the root) to most specific information and define the parent-child relationship between the levels in a hierarchy.

You can define hierarchies where each level rolls up to the previous level in the dimension or you can define hierarchies that skip one or multiple levels.

Part III

Physical Design

This section deals with physical design in a data warehouse.

It contains the following chapters:

- Overview of Physical Design
- Hardware and I/O
- Parallelism and Partitioning
- Indexes
- Constraints
- Materialized Views
- Dimensions

3

Overview of Physical Design

This chapter describes physical design in a data warehousing environment, and includes the following:

- Moving from Logical to Physical Design
- Physical Design

Moving from Logical to Physical Design

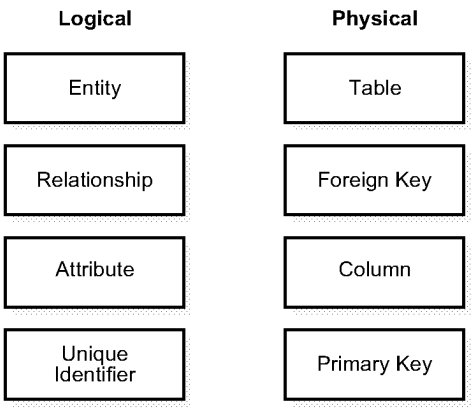
In a sense, logical design is what you draw with a pencil before building your warehouse and physical design is when you create the database with SQL statements.

During the physical design process, you convert the data gathered during the logical design phase into a description of the physical database, including tables and constraints. Physical design decisions, such as the type of index or partitioning have a large impact on query performance. See Chapter 6, "Indexes" for further information regarding indexes. See Chapter 5, "Parallelism and Partitioning" for further information regarding partitioning.

Logical models use fully normalized entities. The entities are linked together using relationships. Attributes are used to describe the entities. The UID distinguishes between one instance of an entity and another.

A graphical way of looking at the differences between logical and physical designs is in Figure 3–1:

Figure 3–1 Logical Design Compared with Physical Design



Physical Design

Physical design is where you translate the expected schemas into actual database structures. At this time, you have to map:

- Entities to Tables
- Relationships to Foreign Keys
- Attributes to Columns
- Primary Unique Identifiers to the Primary Key
- Unique Identifiers to Unique Keys

You will have to decide whether to use a one-to-one mapping as well.

Physical Design Structures

Translating your schemas into actual database structures requires creating the following:

- Tablespaces
- Partitions
- Indexes
- Constraints

Tablespaces

Tablespaces need to be separated by differences. For example, tables should be separated from their indexes and small tables should be separated from large tables. See Chapter 4, "Hardware and I/O", for further information regarding tablespaces.

Partitions

Partitioning large tables improves performance because each partitioned piece is more manageable. Typically, you partition based on transaction dates in a data warehouse. For example, each month. This month's worth of data can be assigned its own partition. See Chapter 5, "Parallelism and Partitioning", for further details.

Indexes

Data warehouses' indexes resemble OLTP indexes. An important point is that bitmap indexes are quite common. See Chapter 6, "Indexes", for further information.

Constraints

Constraints are somewhat different in data warehouses than in OLTP environments because data integrity is reasonably ensured due to the limited sources of data and because you can check the data integrity of large files for batch loads. Not null constraints are particularly common in data warehouses. See Chapter 7, "Constraints", for further details.